FINAL TECHNICAL REPORT TO

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH

by

Jeffery L. Kennington
Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275-0122
jlk@seas.smu.edu
(214)-768-3278

for

INTEGER NETWORKS WITH SIDE CONSTRAINTS: ALGORITHMS AND APPLICATIONS

18 August 1994

F49620-93-1-0091

SMU # 5-25154

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

Dist: A

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Unrestricted A
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 94 0581
6a. NAME OF PERFORMING ORGANIZATION SMU / 6b. OFFICE SYMBOL (If applicable) CSE	7a. NAME OF MONITORING ORGANIZATION AFOSR/NM
6c. ADDRESS (City, State, and ZIP Code) Dallas, TX 75275-0122	7b. ADDRESS (City, State, and ZIP Code) 110 Duncan Ave, Suite 100 Bolling AFB, DC 20332-0001
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR / 8b. OFFICE SYMBOL (If applicable) NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F49620-93-1-0091
8c. ADDRESS (City, State, and ZIP Code) 110 Duncan Ave, Suite 100 Bolling AFB, DC 20332-0001	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. 2304 / TASK NO. DS / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Integer Networks

12. PERSONAL AUTHOR(S) Jeffery L. Kennington

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 1/1/93 TO 30/5/94	94 Aug 18	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	networks, integer programming, optimization

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Many of the routing and scheduling problems which arise at the Air Mobility Command can be modelled as constrained integer networks. The network part is associated with the routing and distribution network flown by the the Command and the side constraints arise when that aircraft capacity must be shared by different commodities or some type of budget restriction must be enforced. The work presented here reports on the progress in solving this type of mathematical program.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION U	
22a. NAME OF RESPONSIBLE INDIVIDUAL Jeffery L. Kennington	22b. TELEPHONE (Include Area Code) 214/768-3278	22c. OFFICE SYMBOL SEAS

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Table of Contents

I. Statement of Work

The Air Mobility Command at Scott Air Force Base has a group of operations research analysts who develop and run mathematical programming models for the Command. Two of their most famous models are the **Patient Evacuation Model** and the **LOGAIR Model.** The problem generator for the **Patient Evacuation Model** has been placed in the public domain and all of the major mathematical programming software groups have tested their software on these models.

Both of these Air Force models involve a large network with additional side constraints. For one model the side constraints describe the capacity of an aircraft which is shared by patients having different injury types. A burn victim may be destined for the burn center in San Antonio where as a soldier with a head injury may be enroute to the Mayo Clinic. For the other model, the side constraints enforce aircraft capacity for cargo sharing the same aircraft but having different origins. That is, cargo that originated at Tinker AFB and cargo that originated at Wright–Patterson AFB may end up on the same aircraft enroute to England, but this cargo must maintain its own identity while on this aircraft so that it will arrive at the proper destination. This is handled by separate networks for each origin node which are linked by mutual capacity constraints. In addition, the clients frequently need integer answers.

CPLEX 3.0 has excellent capabilities, but there are many problems in this class that cause great difficulty for even CPLEX. The author recently ran one of these models for over 10 hours of cpu time on a Dec 5000/260 without obtaining a confirmed optimal solution. It appears, at present, that the only hope for developing robust software for some of these models is to exploit the underlying network structure.

There are two manuscripts presented in this final report. The first concerns a special algorithm and software implementation for the constrained assignment problem. The second fills in a gap in the literature regarding pivot agenda algorithms when the input matrix is singular. Both of these papers are steps in our long term quest to solve the integer constrained network problem.

II. A Branch–and–Bound Algorithm

The constrained assignment problem is to determine a least cost assignment of m men to n jobs such that an additional set of linear constraints is satisfied. This model is a special case of the integer network model with side constraints which in turn is a special case of the binary linear program. This problem is a member of the class NP–Hard and it is well–known that practical problems in this class are intractable. Air Force problems related to the assignment of pilots to schedules and the assignment of aircraft tail numbers to routes can be modelled as constrained assignment problems.

The constraint matrix for the pure network problem without the side constraints has this wonderful property of being totally unimodular. Hence, every basis has determinate equal +1 or −1, every basis is triangular, and every extreme point has integer components. This problem is a member of the class P and is relatively easy to solve. By appending only a single side constraint, the unimodularity property is lost, bases are not triangular, extreme points may not be integer, and the problem is a member of the class NP–Hard. Unfortunately, almost all real–world problems have one or more side constraints. For Air Force models, it is common to have some type of aircraft capacity or budget constraint.

The objective of this study was to develop and empirically evaluate a new algorithm for this model. The algorithm relies on the branch–and–bound strategy and is designed for problems having a large network and a limited number of side–constraints. It exploits an algorithm that we developed earlier for the assignment problem having a single side constraint. This work uses an excellent assignment code that our research team developed and handles the side constraint via the use of Lagrangean relaxation.

The paper resulting from this study appears in Appendix A of this report. It has been submitted for publication and is currently under review.

III. Recovery from Numerical Instability

In my opinion, the best technique available for solving constrained networks is to use a simplex based algorithm in which the basis is partitioned into two parts, one part associated with the network constraints, and one part associated with the side constraints. The component associated with the network part can be maintained as a rooted spanning tree and all operations involving the inverse of this component can be executed using specialized labeling algorithms. Another component, corresponding to the side constraints is called the working basis. It is the inverse of this working matrix which is needed for the operations required by the simplex method.

In our system, the inverse of this working basis is maintained in factored form and every pivot involves the addition of either one or two new factors to the eta file. Periodically, say every 50 to 100 pivots, the working basis is reinverted and a new eta file is developed. The new eta file is smaller than the old one and much of the round–off error which has been introduced during the previous pivots will have been eliminated.

The first step in the procedure to obtain a new factorization is to determine a permutation of the rows and columns so that the sparsity property of this matrix will be maintained in the factorization of its inverse. In the literature, the permutation of the basis is known as the pivot agenda and there are several algorithms for obtaining a good pivot agenda. All pivot agenda algorithms assume that the input matrix is nonsingular.

In our work with specialized partitioning methods for networks with side constraints, we discovered that due to the nature of the side constraints a singular input matrix would eventually be presented to the pivot agenda algorithm. When this occurs, all of the pivot agenda algorithms, of which we are aware, fail. The objective of our investigation was to present recovery procedures, using a variation of the Hellerman–Rarick P3 algorithm, for the case in which the input matrix is singular. The results of our investigation are presented in Appendix B of this document which has been submitted for publication and is currently under review.

Technical Report 93-CSE-49

A BRANCH-AND-BOUND ALGORITHM FOR THE CONSTRAINED ASSIGNMENT PROBLEM

Jeffery L. Kennington
(214)768-3278 jlk@seas.smu.edu
Farin Mohammadi
(214)768-1476 fam@seas.smu.edu

Department of Computer Science and Engineering
School of Engineering and Applied Science
Southern Methodist University
Dallas, Texas 75275-0122

November 1993

Comments and criticisms from interested readers are cordially invited.

ABSTRACT

This manuscript presents a branch-and-bound algorithm to obtain a near optimal solution for the constrained assignment problem in which there are only a few side constraints. At each node of the branch-and-bound tree a lower bound is obtained by solving a singly constrained assignment problem. If needed, Lagrangean relaxation theory is applied in an attempt to improve this lower bound. A specialized branching rule is developed which exploits the requirement that every man be assigned to some job. A software implementation of the algorithm has been tested on problems with five side constraints and up to 75,000 binary variables. Solutions guaranteed to be within 10% of an optimum were obtained for these 75,000 variable problems in from two to twenty minutes of CPU time on a Dec Alpha workstation. We believe that this is the current best algorithm and software implementation for the constrained assignment problem having a limited number of side constraints. The behavior of the branch-and-bound algorithm for various problem characteristics was also studied. This included the tightness of the side constraints, the stopping criteria, and the effect when the problems are unbalanced having more jobs than men.

ACKNOWLEDGMENT

This research was supported in part by the Air Force Office of Scientific Research under Contract Number AFOSR F49620-93-1-0091, and the Office of Naval Research under Contract Number N00014-91-J-1234.

I. INTRODUCTION

The constrained assignment problem is to determine a least cost assignment of m men to n jobs such that an additional set of constraints is satisfied. This model is a binary linear program and may be stated mathematically as follows:

$$\text{minimize} \quad \sum_{(i,j) \in A} c_{ij}x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{j:(i,j) \in A} x_{ij} = 1 \ , \quad i = 1, \ldots, m \tag{2}$$

$$\sum_{i:(i,j) \in A} x_{ij} \leq 1 \ , \quad j = 1, \ldots, n \tag{3}$$

$$x_{ij} \in \{0,1\}, \text{ all } (i,j) \in A \tag{4}$$

$$\sum_{(i,j) \in A} d_{ij}^k x_{ij} \leq r^k \ , \quad k = 1, \ldots, s \tag{5}$$

where $x_{ij} = 1$ implies that man i is assigned to job j at cost of c_{ij}, d_{ij}^k denotes the coefficient of x_{ij} in the kth side constraint, r^k denotes the right-hand-side for the kth side constraint, and A is the set corresponding to the feasible assignments. Note that the problem allows for more jobs than men. Many practical problems have this feature. It also allows for the case in which the number of men exceeds the number of jobs. For this case, one simply reverses the definition of men and jobs.

Since (1)-(5) is a binary linear program, all the literature on integer programming applies (see Geoffrion and Marsten [5], Salkin [12], Parker and Rardin [11], Nemhauser and Wolsey [10]). In practice most integer programming models are either solved as a linear program and the solutions are rounded using some heuristic or branch-and-bound is used in an attempt to obtain a solution within a prespecified tolerance.

A special case in which the side constraints have the generalized upper bound (GUB) structure has been studied by Ali, Kennington, and Liang [2]. A relaxation/ decomposition procedure that involves solving a series of pure assignment problems is used successfully. Ball, Derigs, Hilbrand, and Metz [3] also present an algorithm for the matching problem with generalized upper bound side constraints.

Another special case for s=1 and m=n has been studied by Gupta and Sharma [6], Aggarwal [1], Mazzola and Neebe [9], Bryson [4], Kennington and Mohammadi [7,8]. The only specialized algorithm for (1)-(5) is the two phase procedure of Mazzola and Neebe [9]. The first phase uses subgradient optimization to obtain an advanced start for the branch-and-bound method used in the second phase.

The objective of this study was to develop a new branch-and-bound algorithm to solve the constrained assignment problem and to provide an empirical analysis of this algorithm on a variety of assignment problems having only a few side constraints. All empirical analysis was performed on problems having five side constraints.

II. THE BRANCH-AND-BOUND ALGORITHM

The branch-and-bound method can be viewed as a divide and conquer strategy. If a problem cannot be solved, then it is partitioned into several smaller problems. The best of the solutions to the smaller problems will be the solution to the original problem.

Consider the problem $P(S) \equiv \min\{ cx: x \in S\}$. Using the terminology of Geoffrion and Marsten [4] let $v[P(S)]$ denote the optimal objective function value for the problem $P(S)$. Let x^* denote an incumbent for $P(S)$ with objective value of v^*. Let $\bar{P}(S)$ denote a relaxation of $P(S)$ and let CL denote the candidate list. The generic branch-and-bound algorithm may be stated as follows:

Input:

1. The problem, $P(S)$.

Output:

1. The solution vector, x^*.
2. The objective value corresponding to x^*, v^*.($v^* = \infty$ implies that $S = \Phi$.)

```
Procedure BAB;
  Begin
    initialize:
      CL:= {P(S)}, v*:= ∞;
    while CL ≠ Φ do
      comment: select a candidate problem for analysis.
      select P(U) ∈ CL, CL:= CL \ {P(U)};
      if $\bar{P}(U)$ has a feasible solution, then
```

```
            if $v[\overline{P}(U)] < v^*$, then
              let $\overline{x}$ be an optimum for $\overline{P}(U)$;
              if $\overline{x} \in S$, then
                $x^* := \overline{x}$, $v^* := c\overline{x}$;
              else
                apply a heuristic to $\overline{x}$ in an attempt to create $\dot{x}$ such that $\dot{x} \in S$ and
                $c\dot{x} < v^*$;
                if successful, then $x^* := \dot{x}$, $v^* := c\dot{x}$;
                comment: branching
                create $U_1, U_2, \ldots, U_p$ such that $U_1 \cup U_2 \cup \ldots \cup U_p = U$ and $U_i \cap U_j = \Phi$
                for all $i \neq j \in \{1, 2, \ldots, p\}$;
                CL:= $CL \cup \{P(U_1), P(U_2), \ldots, P(U_p)\}$;
              end if
            end if
          end if
        end while
      end.
```

III. AN EXACT ALGORITHM FOR THE CONSTRAINED ASSIGNMENT PROBLEM

In this section we present a specialized branch-and-bound algorithm for the constrained assignment problem. The constrained assignment problem is a binary problem therefore at each node of the branch-and-bound tree either an assignment is prohibited by fixing the corresponding variable to zero or a man is permanently assigned to a job by fixing the corresponding variable to one.

3.1 The Relaxation

Let D denote the matrix corresponding to the coefficients in (5), x denote the vector corresponding to the binary decision variables, c denote the vector of costs, and r denote the vector of right-hand-side values for the side constraints. Then the constrained assignment problem may be denoted as P(S) where $S = \{x: (2), (3), (4), (5)\}$. Let $\underline{1}$ denote a vector of 1's and $\overline{S} = \{x: (2), (3), (4), \underline{1}Dx \leq \underline{1}r\}$. Then $P(\overline{S})$ is a valid relaxation for P(S). Application of the algorithm in [7] to solve the singly constrained assignment problem will yeild a lower bound for P(S), the optimal Lagrangean multiplier corresponding to the constraint $\underline{1}Dx \leq \underline{1}r$, and $\overline{x} \in \overline{S}$. If $\overline{x} \in S$, then $c\overline{x}$ is an upper bound for P(S).

Let $\dot{S} = \{x: (2), (3), (4)\}$. Recall that a Lagrangean dual for P(S) is the problem max $\{L(a) : a \geq 0\}$ where $L(a) = \min \{cx + a(Dx-r): x \in \dot{S}\}$. We use the optimal Lagrangean multiplier and $\overline{x}$ from the singly constrained algorithm to form an advanced starting value for a. Let y denote the solution for $L(a)$. Then $z = Dy-r$ is used to modify a for successive steps. A limited number of these steps will be performed in this algorithm.

3.2 The Branching Rule

Consider any node in the branch-and-bound tree. If the relaxation, $P(\overline{S})$ has no feasible solution, then this node may be fathomed. Otherwise, an assignment will be used to create the branches as illustrated in Figure 1.

Figure 1 here

For a given node in the branch-and-bound tree let $F^1 = \{(i,j): \overline{x}_{ij} = 1\}$ and $F^0 = \{(i,j): \overline{x}_{ij} = 0\}$. Let $\overline{U} = \{x \in \overline{S}: x_{ij} = 1$ for all $(i,j) \in F^1$ and $x_{ij} = 0$ for all $(i,j) \in F^0\}$. The relaxation solved at each node in the branch-and-bound tree is $P(\overline{U})$ which is a singly constrained assignment problem with some assignments fixed. Let $\overline{x} \in \overline{U}$, $T = \{(i,j) : \overline{x}_{ij} = 1, (i,j) \notin F^1\}$, and $t = |T|$. Consider the t+1 subsets of $\overline{U}$ $(\overline{U}_1, \overline{U}_2, \ldots, \overline{U}_{t+1})$ created in the following manner:

$\overline{U}_1 = \{ x \in \overline{U}: x_{i_1j_1} = 0, (i_1,j_1) \in T\}, T_1 = T\setminus\{(i_1,j_1)\};$

$\overline{U}_2 = \{ x \in \overline{U}: x_{i_1j_1} = 1, x_{i_2j_2} = 0, (i_2,j_2) \in T_1\}, T_2 = T_1\setminus\{(i_2,j_2)\};$

$\overline{U}_3 = \{ x \in \overline{U}: x_{i_1j_1} = 1, x_{i_2j_2} = 1, x_{i_3j_3} = 0, (i_3,j_3) \in T_2\}, T_3 = T_2\setminus\{(i_3,j_3)\};$

.

.

.

$\overline{U}_t = \{ x \in \overline{U}: x_{i_1j_1} = 1, \ldots, x_{i_{t-1}j_{t-1}} = 1, x_{i_tj_t} = 0, (i_t, j_t) \in T_{t-1}\};$

$\overline{U}_{t+1} = \{ x \in \overline{U}: x_{ij} = 1$ for all $(i,j) \in T\}$. Note that, $\overline{U} = \overline{U}_1 \cup \overline{U}_2 \cup \ldots \cup \overline{U}_{t+1}$ and $\overline{U}_i \cap \overline{U}_j = \Phi$ for all $i \neq j$.

Consider a node in the branch-and-bound tree having f_1 edges fixed at 1. Then branching from this node will produce $t+1 = n-f_1+1$ new candidate problems. From Figure 1 it can be seen that the last node (i.e., U_{t+1}) need not be created since it was examined at the parent node. Therefore, each branching produces t candidate problems.

3.3 The Candidate List

For our implementation of the branch-and-bound algorithm, ASSIGN+1 [7] will be applied to $P(\overline{U}_i)$ and the results placed in the candidate list (i.e., a problem is solved before it is placed in the candidate list). The motivation for placing solved problem in the candidate list is that the solution for $P(\overline{U}_i)$ can be easily modified to obtain an advanced starting solution for $P(\overline{U}_{i+1})$. Therefore solving the sequence of problems $P(\overline{U}_1)$, $P(\overline{U}_2)$, ..., $P(\overline{U}_t)$ should require only a moderate amount of computational effort. Hence, each entry in the CL consists of the five tuple $(F^0, F^1, \overline{x}, \overline{\beta}, u)$ where $\overline{x} \in \overline{U}_i$, $\overline{\beta} \leq v[P(\overline{U}_i)]$, and u the optimal Lagrangean dual for the singly constrained problem.

3.4 The Fathoming Rules

At any node p of the branch-and-bound tree let U, F^1, and F^0 be as defined in Section 3.2. Let $M = \{1, 2, 3, \ldots, m\} \setminus \{i: (i,j) \in F^1\}$, then the following rules may be used to fathom a node. If

$$\sum_{(i,j) \in F^1} d_{ij}^k + \sum_{i \in M} \min(d_{ij}^k \ : \ (i,j) \in A \setminus F^0) > r^k$$

for any k, then node p can be fathomed. That is, no selection of the free variables will satisfy the kth side constraint. If

$$\sum_{(i,j) \in F^1} c_{ij} + \sum_{i \in M} \min(c_{ij} \ : \ (i,j) \in A \setminus F^0) > v^*$$

then node p can be fathomed. That is, no selection of the free variables will result in a solution superior to the incumbent.

If $\min \{\underline{1}Dx: x \in U\} > \underline{1}r$, then node p can be fathomed. That is, no selection of the free variables will satisfy all side constraints simultaneously. Let β be the best lower bound obtained for node p and ϵ be the termination tolerance. We will fathom node p if $v^* - \beta \leq \epsilon\beta$. Using this rule with $\epsilon = 0.1$ results in a solution from the branch-and-bound procedure guaranteed to be within 10% of the optimum and $\epsilon = 0.01$ produces a solution within 1% of an optimum.

3.5 The Algorithm

In this section we combine the information presented in Sections 3.1-3.4 to construct the ASSIGN+s algorithm.

Input:

1. The cost vector, c.
2. The feasible region S.
3. The set of (man, job) pairs corresponding to eligible assignments, A.
4. Termination tolerance, ϵ.
5. The maximum execution time, tmax.
6. The maximum number of Lagrangean relaxations to be solved at each node, limit.

Output:

1. The solution vector, x^*.
2. The objective value corresponding to x^*, v^*. ($v^* = \infty$ implies that the problem is infeasible.)

```
Procedure ASSIGN+s;
  Begin
    initialize:
      comment: Node 1 in the Branch-and-Bound tree.
      $v^* := \infty$, $F^0 := \Phi$, $F^1 := \Phi$;
    ASSIGNP1($P(\overline{S})$, $\overline{x}$, $\beta$, u);
    if $\beta = -\infty$ then terminate;
    if $\overline{x} \in S$ then $x^* := \overline{x}$, $v^* := c\overline{x}$;
    else LAGRANGE($\overline{x}$, $\beta$, u);
    comment: Tolerance test for fathoming.
    if $v^* - \beta \leq \epsilon\beta$ then terminate;
    CL:= {($F^0$,$F^1$, $\overline{x}$, $\beta$, u)};
    while CL $\neq \Phi$ do
      SELECT A PROBLEM($F^0$,$F^1$, $\overline{x}$, $\beta$, u);
      BRANCH(CL, $F^0$,$F^1$, $\overline{x}$, $\beta$, u);
    end while
  end.
procedure ASSIGNP1($P(\overline{U})$, $\overline{x}$, $\beta$, u);
  Begin
    initialize:
      $\beta := -\infty$;
    apply the ASSIGN+1 algorithm (Kennington, Mohammadi [1991]) to $P(\overline{U})$;
    if $P(\overline{U})$ has a feasible solution then
      let $\overline{x}$ be the best feasible solution found for $P(\overline{U})$;
      let $\beta$ be the best lower bound found for $P(\overline{U})$;
      let u be the optimal Lagrangean dual for the singly constrained problem;
    end if
  end.
```

```
procedure SELECT A PROBLEM ($F^0, F^1, \bar{x}, \beta, u$);
  Begin
    select ($F^0, F^1, \bar{x}, \beta, u$) $\in$ CL, CL:= CL \ ($F^0, F^1, \bar{x}, \beta, u$);
  end.
procedure BRANCH(CL, $F^0, F^1, \bar{x}, \beta, u$);
  Begin
    initialize:
      $t := 0$, $G := \{(i,j): \bar{x}_{ij} = 1, (i,j) \notin F^1\}$;
      $M := \{1, 2, 3, \ldots, m\} \setminus \{i: (i,j) \in F^1\}$;
    while $G \neq \Phi$ do
      comment: Fix a variable at zero.
      let $(i_1,j_1) \in G$, $G := G \setminus \{(i_1,j_1)\}$, $F^0 := F^0 \cup \{(i_1,j_1)\}$;
      ASSIGNP1($P(\bar{U})$, $\bar{x}$, $\beta$, u);
      if $\beta \neq -\infty$ then
        if $\bar{x} \in S$ and $c\bar{x} < v^*$ then $x^* := \bar{x}$ , $v^* := c\bar{x}$;
        else LAGRANGE($\bar{x}$ , $\beta$, u);
        comment: Tolerance test for fathoming.
        if $v^* - \beta > \epsilon\beta$ then CL:= CL $\cup$ $\{(F^0, F^1, \bar{x}, \beta, u)\}$;
        comment: Permanently assign man $i_1$ to job $j_1$.
        $M := M \setminus \{i_1\}$, $F^1 := F^1 \cup \{(i_1,j_1)\}$;
        $F^0 := F^0 \cup \{(i_1,j) : (i_1,j) \in A \text{ for all } j\} \cup \{(i,j_1) : (i,j_1) \in A \text{ for all } i\} \setminus \{(i_1,j_1)\}$;
        comment: Assignment polytope feasibility tests.

        if $\sum_{(i,j) \in F^1} c_{ij} + \sum_{i \in M} \min(c_{ij} : (i,j) \in A) > v^*$ then return;
        for k=1,...,s
          if $\sum_{(i,j) \in F^1} d_{ij}^k + \sum_{i \in M} \min(d_{ij}^k : (i,j) \in A) > r^k$ then return;
        end for
      end if
    end while
  end.
Procedure LAGRANGE($\bar{x}$, $\beta$, u);
  Begin
    initialize:
      $\alpha := u\underline{1}$, $y := \bar{x}$, $t := 1$;
    while ($v^* - \beta > \epsilon\beta$ and $t \leq$ limit) do
      $z := Dy - r$;
      for i=1,...,s
        if $z_k > 0$, then $\alpha_k := 1.25\alpha_k$;
        if $z_k < 0$, then $\alpha_k := 0.75\alpha_k$;
      end for
      let y be a solution for $L(\alpha)$ and $\beta := \max\{\beta, v[L(\alpha)]\}$;
      if $y \in S$ and $cy < v^*$ then $x^* := y$, $v^* := cy$;
      $t := t+1$;
    end while
end.
```

This branch-and-bound algorithm exploits the structure of the model (1)-(5) in several ways. Permanent assignment of a man to a job implies that all other variables involving this man and job may be fixed at zero. The relaxation was a singly constrained assignment problem for which near optimal integer solutions can be obtained using the results in Kennington and Mohammadi [8]. Special fathoming rules were developed which were based upon the assignment polytope.

IV. EMPIRICAL ANALYSIS

The algorithm ASSIGN+s has been implemented in software and empirically analyzed on an Alpha workstation by Digital Equipment Corporation. The code is written in Fortran and uses ASSIGN+1 (see Kennington and Mohammadi [7]) to solve the singly constrained assignment problems. ASSIGN+1 is an implementation of the Lagrangean relaxation algorithm for sparse singly constrained assignment problems.

We developed a test problem generator with the following inputs: (i) the number of men, (ii) number of jobs for each man, (iii) the maximum cost, $\bar{c}$, (iv) the number of side constraints, s, and (v) the side constraint multiplier, k. Both the costs and the side constraint coefficients are uniformly distributed over the range $[0, \bar{c})$. We randomly generate a feasible assignment, $\bar{x}$ and set the right-hand-side of the side constraints, r, to $kD\bar{x}$. Obviously, as k becomes smaller, the feasible region becomes smaller and for sufficiently small k {x: (2), (3), (4), and (5)} is usually empty.

The generator was used to generate two sets of 400x400 problems described in Table 1. As Table 1 indicates, the problems generally become more difficult as k becomes smaller and for k small enough the feasible region is empty. For all runs, the stopping criteria used is ϵ=10% and the % deviation reported in column 8 gives a guarantee on the deviation from optimality. All times are the CPU time and exclude the time for both input and output. The run with problem number 2 having k=0.4 was terminated after the candidate list grew to 25,000 entries. As we expected, there exists problems which cannot be solved in a reasonable amount of time and storage using this approach. Tightly constrained problems having 48,000 binary variables definitely stretches the capability of this software implementation of our branch-and-bound algorithm.

Table 1 here

Tables 2 and 3 give our empirical results with 30 randomly generated assignment problems with various sizes all having five side constraints. For all of the problems tested we were able to find a solution guaranteed to be within 10% of an optimal solution. The six smallest problems have 3,000 binary variables. Five of these were solved in less than two minutes each and one required about six and one-half minutes. The six largest problems had 75,000 binary variables and were all solved in less than twenty one minutes each. The most difficult problems (300x300) have 27,000 binary variables. Two of these six problems required eighty minutes to solve. These two difficult problems also had very tight side constraints.

Tables 2 and 3 here

This work was motivated by models for assigning sailors to ships and for this application the number of jobs always exceeds the number of sailors available. Frequently the job list covers a longer period than the list of available sailors which produces a large imbalance in n and m. Tables 4 and 5 present our empirical results from solving 18 unbalanced assignment problems with five side constraints. For the 300x600 problem with $k=0.6$ presented in Table 5, the run was terminated due to candidate size limit. For all other test problems we were able to obtain a solution within 10% of an optimum.

Tables 4 and 5 here

For all test problems we search for a solution within 10% of an optima. To study the effect of the tolerance value on the performance of the algorithm we solved two 200x200 and two 200x400 problems with different tolerance values. Figure 2 indicates that, as expected, a decrease in the tolerance value leads to an increase in the execution time. For all four problems, a point was reached in which a slight decrease in the tolerance resulted in a large increase in the solution time.

Figure 2 here

V. SUMMARY AND CONCLUSIONS

We have presented a branch-and-bound algorithm for the constrained assignment problem. The algorithm is applicable for both balanced and unbalanced assignment problems having inequality side constraints. The algorithm uses a specialized branching rule that exploits the underlying structure of the problem. Bounds are obtained by solving a singly constrained assignment problem followed by a few iterations with a Lagrangean relaxation.

We presented empirical results for both balanced and unbalanced problems having five side constraints. For problems having 75,000 binary variables, solutions guaranteed to be within 10% of an optima were obtained in less than twenty one minutes on a Dec Alpha workstation. Our analysis indicated that as the side constraints become tighter the execution time and number of branch-and-bound nodes increases. For one of the 300x300 problems having 27,000 arcs, the execution time increased from about one minute to about eighty minutes as a result of side constraint tightening. Our analysis also indicates that the performance of the algorithm on unbalanced problems is generally better than its performance for the balanced problems with the same number of binary variables. The Navy personnel assignment problems which motivated this study are all unbalanced models.

For problems of this type, having only a few side constraints, we believe that this is the current best algorithm and software implementation available. Solutions guaranteed to be within 10% of an optimum should be obtained for most problems having fewer than five side constraints and fewer than 20,000 arcs. However, as with any other branch-and-bound based procedure, we found difficult problems which required an extraordinary amount of computer time.

VI. REFERENCES

1. V. Aggarwal, "A Lagrangean-Relaxation Method for the Constrained Assignment Problem," *Computers and Operations Research* vol. 12 pp. 97–106, 1985.

2. I. Ali, J. Kennington, and T. Liang, "Assignment with En Route Training of Navy Personnel," *Naval Research Logistics Quarterly* vol. 40 pp. 581–592, 1993.

3. M. Ball, U. Derigs, C. Hilbrand, and A. Metz, "Matching Problems with Generalized Upper Bound Side Constraints," *Networks* vol. 20 pp. 703–721, 1990.

4. N. Bryson, "Parametric Programming and Lagrangian Relaxation: The Case of the Network Problem with a Single Side-Constraint," *Computers and Operations Research* vol. 18 pp. 129–140, 1991.

5. A. Geoffrion and R. Marsten, "Integer Programming Algorithms: A Framework and State-Of-The-Art Survey," *Management Science* vol. 18 pp. 465–491, 1972.

6. A. Gupta and J. Sharma, "Tree Search Method for Optimal Core Management of Pressurised Water Reactors," *Computers and Operations Research* vol. 8 pp. 263–269, 1981.

7. J. Kennington and F. Mohammadi, "The Singly Constrained Assignment Problem: An AP Basis Approach," Technical Report 93-CSE-25, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275, 1993.

8. J. Kennington and F. Mohammadi, "The Singly Constrained Assignment Problem: A Lagrangean Relaxation Approach," to appear in *Computational Optimization and Applications.*

9. J. Mazzola and A. Neebe, "Resource Constrained Assignment Scheduling," *Operations Research* vol. 34 pp. 560-572, 1986.

10. G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization*, John Wiley and Sons: New York, NY, 1988.

11. R. Parker and R. Rardin, *Discrete Optimization*, Academic Press Incorporated: New York, NY, 1988.

12. H. Salkin, *Integer Programming*, Addison-wesley Publishing Company: Reading, Massachusetts, 1974.

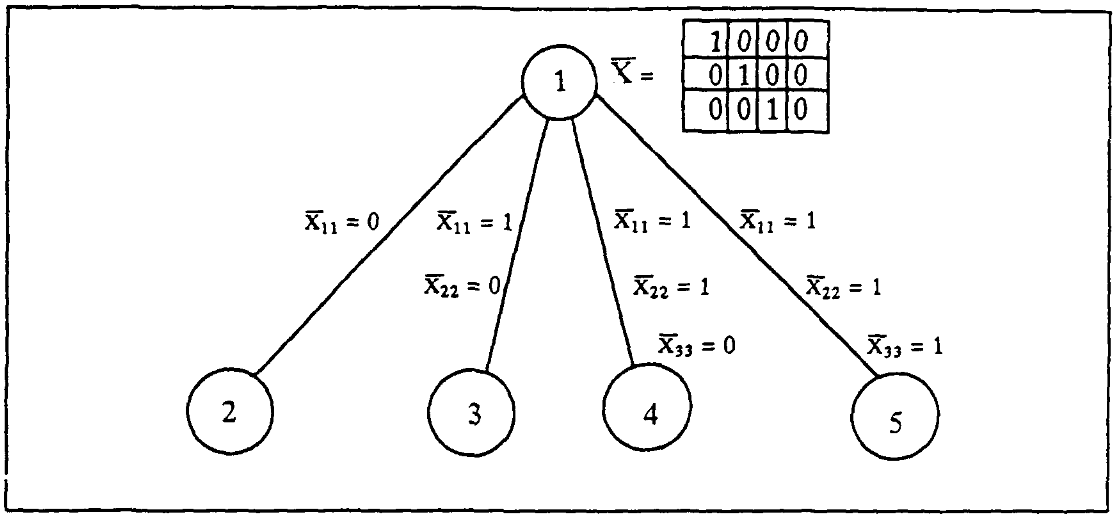
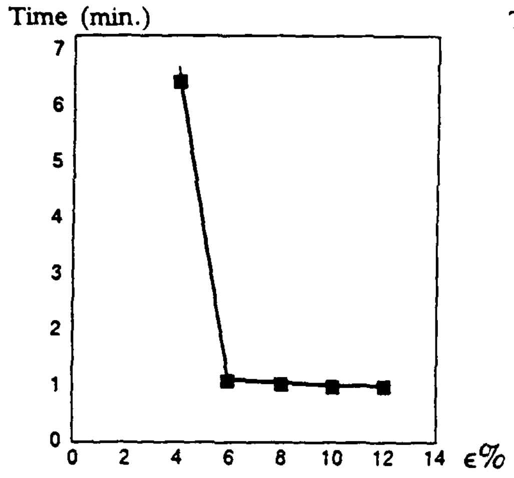
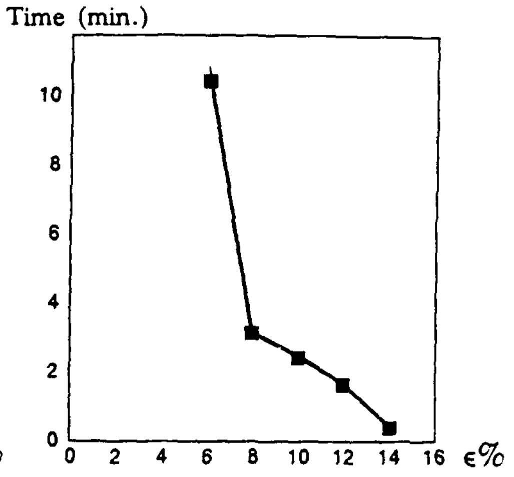


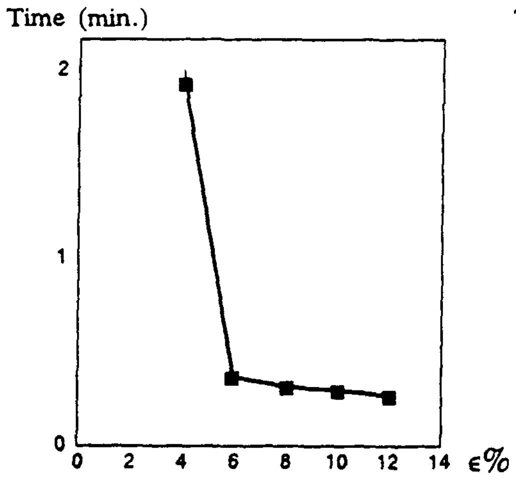
Figure 1. Example of branching rule for a 3x4 problem.



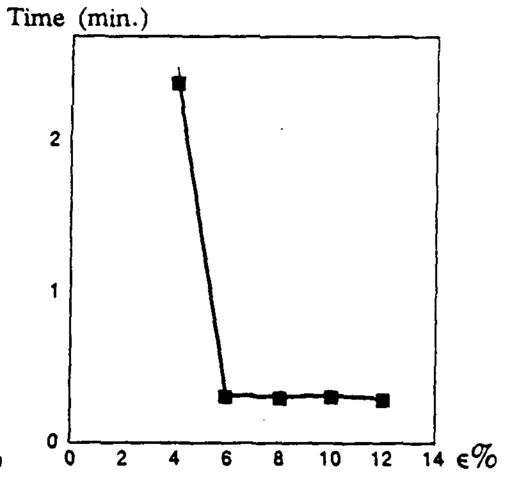
a. Problem #1 (200x200) with 1200 arcs



b. Problem #2 (200x200) with 1200 arcs



c. Problem #3 (200x400) with 1200 arcs



d. Problem #4 (200x400) with 1200 arcs

Figure 2. Plots of time versus the stopping tolerance for four problems.

Table 1. Empirical results from branch-and-bound algorithm for 400x400 assignment problems with five side constraints (48,000 columns and 805 rows).

Prob.	k	# BAB Nodes	# AP's Solved	Time (min.)	LB	UB	% Deviation	Node # of Incumbent
1	1.0	189	216	0.71	5,270	5,309	0.74	13
	0.9	244	1,958	4.74	5,505	5,768	4.78	22
	0.8	274	2,119	3.96	6,999	7,271	3.89	78
	0.7	952	8,288	12.66	10,991	11,725	6.68	859
	0.6	14,142	124,222	166.00	20,820	22,713	9.09	13,943
	0.5	4,214	35,362	45.39	47,446	50,343	6.11	4,157
	0.4	1	2	0.00	problem has no feasible solution			
2	1.0	224	503	1.66	5,370	5,559	3.52	2
	0.9	253	1,801	4.57	5,830	6,132	5.23	34
	0.8	700	5,434	9.26	7,815	8,373	7.13	492
	0.7	3,114	25,786	42.93	12,510	13,649	9.10	2,938
	0.6	3,094	28,694	40.92	23,606	25,188	6.70	2,871
	0.5	3,505	29,642	31.45	50,939	54,690	7.36	3,340
	0.4	25,000[1]	79,160	53.37	148,825	no feasible solution obtained		
	0.3	1	2	0.0	problem has no feasible solution			

[1] terminated due to candidate list size limit.

Table 2. Empirical results with problem set 1 using the branch-and-bound algorithm. (All problems have five side constraints.)

Problem Description			# BAB Nodes	# AP's Solved	Time (min.)	LB	UB	% Deviation	Node # of Incumbent
n x m	Jobs/Man	k							
100x100	30	1.0	215	859	0.06	5,021	5,356	6.67	178
	30	0.8	447	3,730	0.28	6,758	7,090	4.90	404
	30	0.6	11,938	100,799	6.60	18,726	20,599	10.00	5,686
200x200	60	1.0	111	192	0.13	5,131	5,153	0.43	4
	60	0.8	707	5,735	2.04	6,890	7,485	8.62	572
	60	0.6	5,832	52,193	15.78	22,314	24,336	9.95	5779
300x300	90	1.0	184	526	0.82	5,533	5,641	1.95	12
	90	0.8	240	1,892	1.87	7,559	8,006	5.90	78
	90	0.6	12,679	115,382	79.38	21,901	24,062	9.87	12,538
400x400	120	1.0	218	509	1.72	5,160	5,308	2.87	12
	120	0.8	325	2,541	4.63	7,363	7,954	8.03	146
	120	0.6	503	4,782	9.09	21,567	23,614	9.49	143
500x500	150	1.0	265	569	3.37	5,170	5,405	4.55	5
	150	0.8	397	3,421	10.70	7,481	7,833	4.70	139
	150	0.6	610	5,873	20.61	23,385	24,780	5.96	241

Table 3. Empirical results with problem set 2 using the branch-and-bound algorithm. (All problems have five side constraints.)

Problem Description			# BAB Nodes	# AP's Solved	Time (min.)	LB	UB	% Deviation	Node # of Incumbent
n x m	Jobs/Man	k							
100x100	30	1.0	76	413	0.04	4,975	5,383	8.18	4
	30	0.8	1,394	12,757	1.02	7,974	8,598	7.82	1,337
	30	0.6	2,932	25,891	1.86	21,253	22,756	7.07	2,871
200x200	60	1.0	103	196	0.13	5,873	5,928	0.85	6
	60	0.8	1	8	0.00	8,010	8,783	9.64	1
	60	0.6	1,457	13,308	4.03	24,194	25,954	7.27	1,374
300x300	90	1.0	158	290	0.56	5,237	5,280	0.82	10
	90	0.8	490	4,115	4.34	7,658	7,894	3.08	282
	90	0.6	10,813	99,948	79.77	23,985	26,055	8.63	10,659
400x400	120	1.0	250	1,126	2.78	4,892	5,160	5.46	18
	120	0.8	360	3,071	5.26	7,923	8,606	8.61	17
	120	0.6	2,476	23,947	39.17	25,821	28,297	9.59	2,362
500x500	150	1.0	336	1,731	7.50	4,933	5,004	1.42	138
	150	0.8	410	3,385	9.69	7,968	8,514	6.85	134
	150	0.6	630	6,254	19.72	25,838	27,683	7.14	228

Table 4. Empirical results with problem set 3 using the branch-and-bound algorithm. (All problems have five side constraints.)

Problem Description			# BAB Nodes	# AP's Solved	Time (min.)	LB	UB	% Deviation	Node # of Incumbent
n x m	Jobs/Man	k							
100x200	30	1.0	229	1,185	0.07	3,668	3,940	7.39	183
	30	0.8	90	883	0.04	5,384	5,800	7.70	76
	30	0.6	2,726	24,147	0.85	15,302	16,430	7.37	2,724
200x400	60	1.0	44	173	0.06	3,491	3,559	1.95	14
	60	0.8	144	1,556	0.28	5,038	5,224	3.68	142
	60	0.6	248	2,318	0.34	16,990	18,519	9.00	241
300x600	90	1.0	105	417	0.30	3,800	3,937	3.61	2
	90	0.8	271	2,262	0.87	5,297	5,674	7.11	62
	90	0.6	1,200	12,151	4.10	15,809	17,063	7.93	1,174

Table 5. Empirical results with problem set 4 using the branch-and-bound algorithm. (All problems have five side constraints.)

Problem Description			# BAB Nodes	# AP's Solved	Time (min.)	LB	UB	% Deviation	Node # of Incumbent
n x m	Jobs/Man	k							
100x200	30	1.0	47	238	0.02	3,333	3,445	3.36	5
	30	0.8	127	1,247	0.06	5,031	5,490	9.11	74
	30	0.6	9,651	93,433	3.74	14,160	14,650	3.46	9,589
200x400	60	1.0	1	5	0.00	4,332	4,629	6.86	1
	60	0.8	139	1,459	0.28	6,100	6,475	6.14	57
	60	0.6	183	1,933	0.30	18,080	19,303	6.76	131
300x600	90	1.0	101	401	0.24	3,644	3,729	2.50	37
	90	0.8	223	2,177	0.81	5,403	5,551	2.74	162
	90	0.6	25,000	256,241	89.84	17,537	20,611	17.53	15,015

[1] terminated due to candidate size limit

Technical Report 94-CSE-32

Recovery from Numerical Instability During Basis Reinversion

by

Jeffery L. Kennington

and

Riad A. K. Mohamed

Department of Computer Science and Engineering

School of Engineering and Applied Science

Southern Methodist University.

Dallas. Texas 75275

August 1994.

Abstract

All of the preassigned pivot agenda algorithms that extend the Hellerman-Rarick P3 algorithm assume that the input matrix is nonsingular. Due to numerical instability, this assumption may be violated and these algorithms fail. We present a modification of the P3 algorithm which includes a procedure to recover from this type of numerical instability. The recovery procedure is integrated into P3 in such a way that all previous work can be maintained and it reduces the likelihood that additional recovery will be required.

1. Introduction

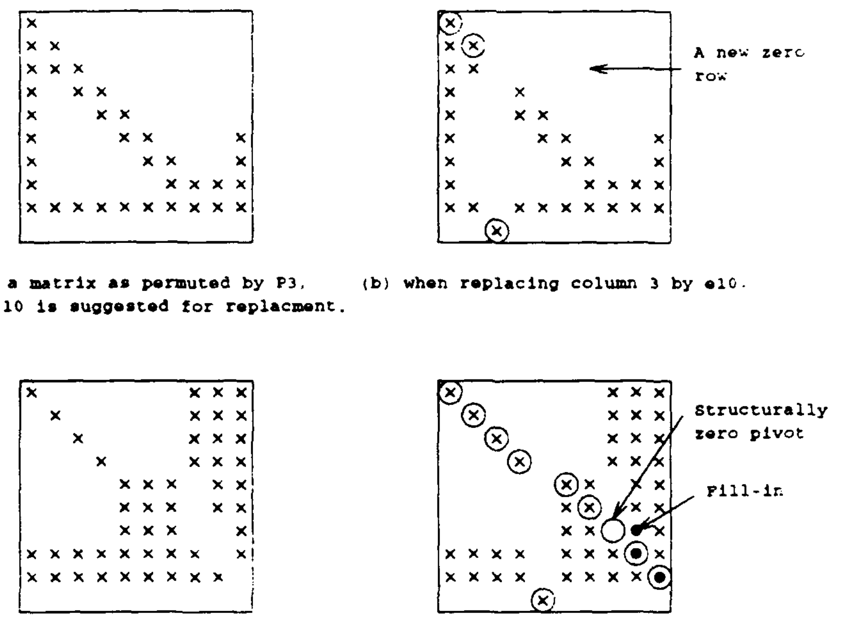
In linear programming systems, an important component is the algorithm for obtaining a new factorization for the inverse of the basis. The first step in the algorithm is to determine a permutation of the rows and columns of the basis, so that the sparsity property of the basis will be maintained in the factorization of it's inverse. In the literature, the permutation of the basis is known as a *pivot agenda*, and there are several algorithms for obtaining a good pivot agenda.

In 1971, Hellerman and Rarick [5] introduced the preassigned pivot procedure (P3) for obtaining such a permutation. In 1972, they added an initial sort that permutes the matrix to lower block triangular form, and then applies a simplified version of the P3 algorithm to each block. They called this algorithm the partitioned preassigned pivot procedure (P4), see [6]. Duff [2], presented a simple algorithm for permuting the matrix so that its diagonal has a minimum number of zeros. Erisman *et al.* [3], discussed the possibility of a structurally zero pivot arising in P4, and suggested a variant that avoids this problem, called the precautionary partitioned preassigned pivot procedure (P5). Arioli *et al.* [1], reported that the P5 algorithm does not address the problems associated with small pivots, and that P4 performed better than P5 when taking into account numerical pivoting. Hattersley and Mackley [4], described a transposed version of P3 which permutes the matrix into an upper triangular form with row spikes extending below the diagonal, and suggested a technique which alternates between the classic and transposed versions of P3 for reducing the build-up of nonzeros during factorization. Sankaran [7], presented some new results on optimal spike configuration, using an ordering heuristic for doubleton columns.

All of these methods assume that the input matrix has at least one nonzero entry in every row. In our work with specialized partitioning methods for networks with side constraints, one maintains the inverse of a working basis, Q, corresponding to the side constraints. Due to the nature of certain types of side constraints and finite precision a Q can be developed having one or more rows, each entry of which is smaller than our zero tolerance. Hence, the usual pivot agenda algorithms fail. For problems in this class, we discovered that numerical instabilities frequently occur. Scaling can alleviate some of these problems, but can not guarantee that this difficulty will not arise. When a row of all zeros (or near zeros) is discovered, one generally replaces some column of Q with an artificial column and the simplex method is continued. This idea is referred to as recovery (i.e. the algorithm recovers from numerical instability).

The objective of this investigation is to present recovery procedures, using a variant of the Hellerman-Rarick P3 algorithm, for the case in which the input matrix has one or more rows of all zeros. Our variation allows decomposing the bump into smaller blocks, and permits interchanges among the unassigned rows to avoid what Erisman *et al.* refer to as a *structurally zero pivot*. Recovery involves replacing some column of the input matrix with an appropriate column of the identity matrix. The difficult part is to determine the column to be replaced so that the current work on the pivot agenda can be retained. An arbitrary selection of the column to be replaced can lead to failure of the P3 algorithms. This failure occurs when it is discovered that there exists a row having all zeros in unassigned columns, or a structurally zero pivot may arise. Both of these conditions will be demonstrated in the example to follow.

Consider the matrix illustrated in Figure 1a, which has all zeros in row 10. Replacing column 3 by e10 (a column with a nonzero in row 10 and zeros elsewhere) and applying the P3 algorithm results in all zeros for unassigned columns in row 3, as illustrated in Figure 1b. This gave us the intuition to confine the column replacement to the lower triangular blocks. However, this does not completely solve the problem. Consider the matrix illustrated in Figure 1c, which also has all zeros in row 10. Replacing column 5 by e10, then applying the P3 algorithm, will introduce a structurally zero pivot at location (7,8), as illustrated in Figure 1d.



(a) a matrix as permuted by P3, column 10 is suggested for replacment.

(b) when replacing column 3 by e10.

(c) a block as permuted by P3, column 10 is suggested for replacment.

(d) when replacing column 5 by e10.

Figure 1: Examples of P3 failure due to incorrect column replacement.

2. A Pivot Agenda

Let **B** be a boolean matrix of order n, and let b_{ij} denote the ij^{th} element of **B**. A pivot agenda for **B** is a permutation of the rows and columns of **B** which yields a matrix having some desirable property. For this work the desirable property is lower triangular form. If this is impossible, then we seek a near lower triangular form. Let $\mathcal{C}$ and $\mathcal{R}$ denote the sequence of n columns and rows in a pivot agenda. That is, the k^{th} column (row) in the permuted matrix is $\mathcal{C}_k(\mathcal{R}_k)$.

A pivot agenda can be developed by using three distinct procedures. One procedure searchs for column singletons and places these columns at the end of the pivot agenda. Another procedure searchs for row singletons and places these rows at the beginning of the pivot agenda. The third procedure determines the permutation for the remaining rows and columns to construct one or more lower triangular blocks. These procedures are repeated until all columns and rows of **B** are permuted to construct a lower triangular block. The pivot agenda algorithm can be described mathematically as follows:

```
Procedure : Pivot_Agenda
Input : B, n.
Output : $\mathcal{C}, \mathcal{R}$.
begin
  $r \leftarrow 1; u \leftarrow n;$
  for ($i = 1, \ldots, n$) do $\mathcal{J}_i \leftarrow \mathcal{I}_i \leftarrow \emptyset;\ \mathcal{C}_i \leftarrow \mathcal{R}_i \leftarrow 0;$
  for ($i = 1, \ldots, n$) do
    for ($j = 1, \ldots, n$) do
      if ($b_{ij} \neq 0$) then $\mathcal{J}_i \leftarrow \mathcal{J}_i \cup \{j\};\ \mathcal{I}_j \leftarrow \mathcal{I}_j \cup \{i\};$      /* Initialization */
    end for
  end for
  $\Xi \leftarrow \{i : 1 \le i \le n, \mathcal{J}_i = \emptyset\};$      /* Record the zero rows for recovery */
  while ($r \le u$) do
    Col_Singl($n, u, r, \mathcal{I}, \mathcal{J}, \mathcal{C}, \mathcal{R}$);
    Row_Singl($n, u, r, \mathcal{I}, \mathcal{J}, \mathcal{C}, \mathcal{R}$);
    Bump_Proc($n, u, r, \mathbf{B}, \mathcal{I}, \mathcal{J}, \mathcal{C}, \mathcal{R}, \Xi$);
  end while
end.
```

3. Column Singletons

We refer to $\mathcal{C}$ and $\mathcal{R}$ as a partial pivot agenda if there exists an index $1 \le k \le n$ such that $\mathcal{C}_k = \mathcal{R}_k = 0$. Let $\mathcal{I}_j = \{i : b_{ij} \neq 0, i \notin \mathcal{R}, 1 \le i \le n\}$. For $j \notin \mathcal{C}$, if $|\mathcal{I}_j| = 1$, then column j is called a *column singleton*. Placing a column singleton and the corresponding row to the right most unassigned position of the pivot agenda (i.e. $\mathcal{C}_u = \mathcal{R}_u = 0$ for the largest u) results in appending one additional lower triangular column. In this procedure we repeat the search for column singletons until all row and columns are assigned to $\mathcal{C}(\mathcal{R})$, or until no column singletons are found. The search for column singletons can be represented mathematically as follow:

Procedure : *Col_Singl*

Input : $u, n, c, \mathcal{I}, \mathcal{J}$.

Output : $u, \mathcal{I}, \mathcal{J}, \mathcal{C}, \mathcal{R}$.

begin

```
  while ($\exists j : |\mathcal{I}_j| = 1, 1 \le j \le n$ & $c \le u$) do
    $\mathcal{C}_u \leftarrow j$; $\mathcal{R}_u \leftarrow i \in \mathcal{I}_j$;
    $\mathcal{I}_k \leftarrow \mathcal{I}_k \setminus \{i\}\ \forall k \in \mathcal{J}_i$;
    $\mathcal{J}_i \leftarrow \emptyset$; $u \leftarrow u - 1$;
  end while
```

end.

4. Row Singletons

Let $\mathcal{C}$ and $\mathcal{R}$ be any partial pivot agenda, and let $\mathcal{J}_i = \{j : b_{ij} \neq 0, j \notin \mathcal{C}, 1 \le j \le n\}$. For $i \notin \mathcal{R}$, if $|\mathcal{J}_i| = 1$, then row i is called a *row singleton*. Placing a row singleton and the corresponding column to the left most unassigned position of the pivot agenda (i.e. $\mathcal{C}_c = \mathcal{R}_c = 0$ for the smallest c) results in appending one additional lower triangular row. In this procedure we repeat the search for row singletons until all row and columns are assigned to $(\mathcal{C}, \mathcal{R})$, or until no row singletons are found. The search for row singletons can be represented mathematically as follow:

```
Procedure : Row_Singl
Input : $n, u, r, \mathcal{I}, \mathcal{J}$.
Output : $r, \mathcal{I}, \mathcal{J}, \mathcal{C}, \mathcal{R}$.
begin
    while$(\exists i : |\mathcal{J}_i| = 1, 1 \leq i \leq n) \& r \leq u$ do
        $\mathcal{R}_r \leftarrow i$; $\mathcal{C}_r \leftarrow j \in \mathcal{J}_i$;
        $\mathcal{J}_k \leftarrow \mathcal{J}_k - \{j\}\ \forall k \in \mathcal{I}_j$;
        $\mathcal{I}_j \leftarrow \emptyset$; $r \leftarrow r + 1$;
    end while
end.
```

Consider the matrix illustrated in Figure 2. Applying procedure *Col_Singl* to this matrix results in placing columns 1 and 2 in the last two columns of the permuted matrix. Applying procedure *Row_Singl* to the permuted matrix results in placing rows 6, 9, 11, and 12 in the first four rows of the permuted matrix. The permuted matrix is illustrated in Figure 3.

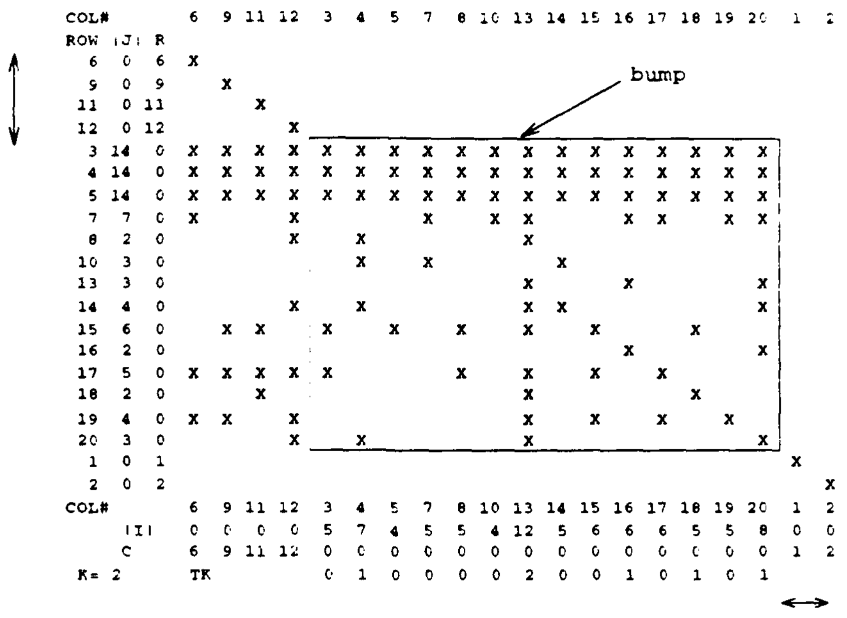
5. The Bump

After applying the *Col_Singl* and *Row_Singl* procedures, the remaining rows, $\{i : 1 \leq i \leq n, i \notin \mathcal{R}\}$, and columns, $\{j : 1 \leq j \leq n, j \notin \mathcal{C}\}$, will either have $|\mathcal{J}_i| > 1$ and $|\mathcal{I}_j| > 1$, or $|\mathcal{J}_i||\mathcal{I}_j| = 0$. For this section, we assume that $|\mathcal{J}_i||\mathcal{J}_i| \neq 0$. These remaining rows and columns will form a nontriangular section of the permuted matrix, called the *bump*.

```
COL#             1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
ROW  |J|   R
   1   1   0     X
   2   1   0         X
   3  18   0             X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
   4  18   0             X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
   5  18   0             X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
   6   1   0                         X
   7   9   0                         X   X           X       X   X           X   X       X   X
   8   3   0                 X                               X   X
   9   1   0                                 X
  10   3   0                 X           X                           X
  11   1   0                                         X
  12   1   0                                             X
  13   3   0                                                 X           X               X
  14   5   0                 X                               X   X   X                   X
  15   8   0             X       X           X   X       X       X       X           X
  16   2   0                                                             X               X
  17   9   0             X           X       X   X       X   X   X       X       X
  18   3   0                                         X       X                   X
  19   7   0                         X           X           X   X       X       X       X
  20   4   0                 X                               X   X                       X
COL#             1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
         |I|     1   1   5   7   4   7   5   5   7   4   7  10  12   5   6   6   6   5   5   8
           C     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Figure 2: The boolean matrix.

We seek a permutation of the bump rows and columns that (i) yields only a few small blocks that extend above the diagonal, (ii) that minimizes the number of nonzero entries that appear above the diagonal in these blocks, and (iii) that confines these entries to a few columns that extend above the diagonal, called the *spikes*. To obtain this permutation, select a column or a set of columns, which will introduce the maximum number of row singletons when temporarily removed from the bump. Let $t_k(j) = |\{i : i \in \mathcal{I}_j, \ 0 < |\mathcal{J}_i| \leq k\}|$, which is called the k^{th} order *tally function* of column j. For a given k, determine t_k for every column in the bump. If the maximum t_k is greater than one, select the corresponding column for temporary removal using the maximum $|\mathcal{I}_j|$ to break ties. If the maximum t_k equals one, only one row will be affected by the temporary removal of this column. For this case, increase k to the minimum $|\mathcal{J}_i|$ greater than k and repeat the process. Let $\mathcal{S}$ denote the sequence of columns which are temporarily removed, (spikes). A column selection method using



```
COL#           6   9  11  12   3   4   5   7   8  10  13  14  15  16  17  18  19  20   1   2
ROW  |J|   R
   6   0   6   X
   9   0   9       X
  11   0  11           X
  12   0  12               X
   3  14   0   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
   4  14   0   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
   5  14   0   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
   7   7   0   X           X               X       X   X           X   X       X   X
   8   2   0               X       X                   X
  10   3   0                       X       X                   X
  13   3   0                                           X           X               X
  14   4   0               X       X                   X   X                       X
  15   6   0       X   X       X       X       X       X       X           X
  16   2   0                                                       X               X
  17   5   0   X   X   X   X   X               X       X       X       X
  18   2   0           X                               X                   X
  19   4   0   X   X       X                           X       X       X       X
  20   3   0               X       X                   X                           X
   1   0   1                                                                           X
   2   0   2                                                                               X
COL#           6   9  11  12   3   4   5   7   8  10  13  14  15  16  17  18  19  20   1   2
     |I|       0   0   0   0   5   7   4   5   5   4  12   5   6   6   6   5   5   8   0   0
       C       6   9  11  12   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   2
  K= 2        TK               0   1   0   0   0   0   2   0   0   1   0   1   0   1
```

Figure 3: The matrix after applying the *Col_Singl* and *Row_Singl* procedures.

the tally function can be described mathematically as follows:

Procedure : *Max_Tally*

Input : $k, n, \mathcal{I}, \mathcal{C}, \mathcal{S}$.

Output : k, s.

```
begin
  $\mathcal{N} \leftarrow \{j : 1 \leq j \leq n,\ j \notin \mathcal{C},\ j \notin \mathcal{S}\};\ m \leftarrow 1;$
  while $(m = 1\ \&\ \mathcal{N} \neq \phi)$ do
    for $(j \in \mathcal{N})$ do $t_k(j) \leftarrow |\{i : i \in \mathcal{I}_j,\ 0 < |\mathcal{J}_i| \leq k\}|;$
    $m \leftarrow \max\{t_k(j) : j \in \mathcal{N}\};$
    if( $m = 1$ )then
      $\mathcal{N} \leftarrow \{j : 1 \leq j \leq n,\ t_k(j) = 1\};$
      $k \leftarrow \min\{|\mathcal{J}_i| : k < |\mathcal{J}_i|,\ 1 \leq i \leq n\};$
    end if
    end while
    $s \leftarrow \text{argmax}\{ T_j : j \in \mathcal{N}, t(j) = m_k \}$;
end.
```

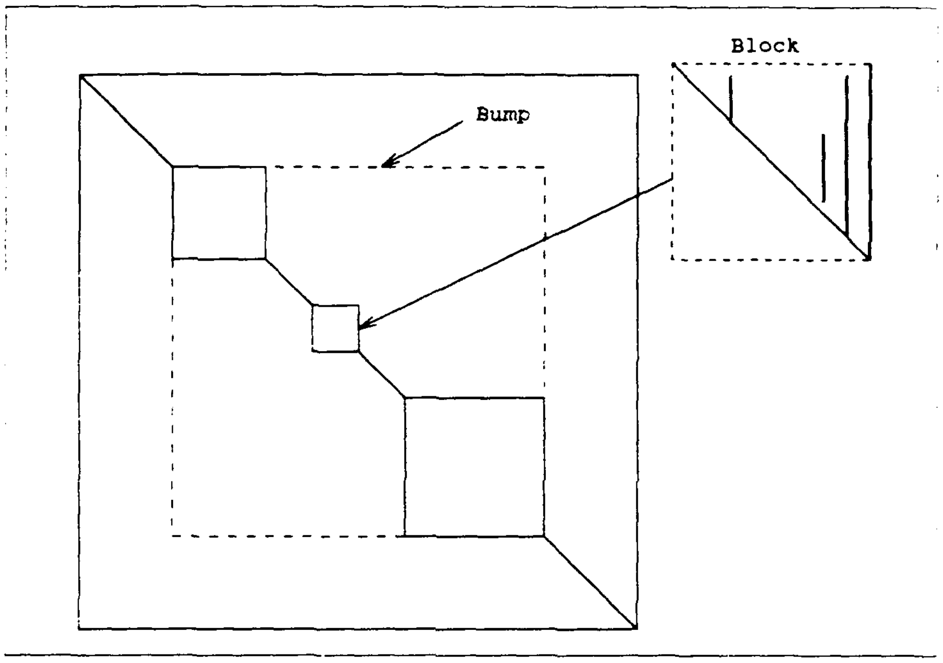


Figure 4: *Block partition of a bump.*

If possible, the bump is decomposed into a small number of blocks connected by lower triangular components. An example of this structure is illustrated in Figure 4. Within a block, only the spikes are permitted to have a zero on the diagonal. The bump processing technique uses the *Max_Tally* procedure for spike selection. It begins by setting k to the minimum $|\mathcal{J}_i|$. A column is selected, using the *Max_Tally* procedure, removed from the bump, and assigned to the spike sequence $\mathcal{S}$. This process is repeated until at least one row singleton is obtained. If the row singleton is unique, the corresponding pivot is placed in $\mathcal{C}_r(\mathcal{R}_r)$. Otherwise k is set to 1, a column, s, is then selected using the *Max_Tally* procedure and placed in $\mathcal{C}_r$. Then one of the row singletons that has a nonzero entry in column s is selected, and placed in $\mathcal{R}_r$. If

$q = t_1(s) > 1$, i.e. the number of row singletons having nonzero entries in column s is greater than one, we have the opportunity to place $(q - 1)$ spikes into the pivot agenda. Including all the spikes into the pivot agenda isolates a block, and decomposes the bump into smaller blocks. The process is repeated for the identification of each block. If **B** has a row of all zeros, or a column of all zeros, i.e. $|\mathcal{I}_j||\mathcal{J}_i| = 0$, then the desired lower triangular structure is not achievable. If this is discovered, a recovery procedure, which is described in the next section, will be invoked. The bump processing can be described mathematically as follows:

Procedure : *Bump_Proc*

Input : $n, u, v, \mathbf{B}, \mathcal{I}, \mathcal{J}, \Xi$.

Output : $u, v, \mathbf{B}, \mathcal{I}, \mathcal{J}, \mathcal{C}, \mathcal{R}, \Xi$.

```
begin
  $t \leftarrow 0; \mathcal{S} \leftarrow \emptyset;$
  $k \leftarrow \min\{|\mathcal{J}_i| : \mathcal{J}_i \neq \emptyset, 1 \le i \le n\};$
  if ($\{i : \mathcal{J}_i \neq \emptyset, 1 \le i \le n\} = \emptyset$ & $\Xi \neq \emptyset$) then $Recover\_1(k, n, u, \mathbf{B}, \mathcal{I}, \mathcal{C}, \mathcal{R}, \Xi);$
  while ($(k > 1 \| t > 0)$ & $v \le u$) do
    while ($k > 1$) do                /* temporarily remove a set of columns */
      $Max\_Tally(k, n, \mathcal{I}, \mathcal{C}, \mathcal{S}, s);$
      $t \leftarrow t + 1;$ $\mathcal{S}_t \leftarrow s;$ $\mathcal{J}_k \leftarrow \mathcal{J}_k \setminus \{s\}$ $\forall k \in \mathcal{I}_s;$ $\mathcal{I}_s \leftarrow \emptyset;$
      $k \leftarrow \min\{|\mathcal{J}_i| : \mathcal{J}_i \neq \emptyset, 1 \le i \le n\};$
    end while
    $\mathcal{F} \leftarrow \{i : |\mathcal{J}_i| = 1, 1 \le i \le n\};$          /* record the row singletons */
    while ($\mathcal{F} \neq \emptyset$ & $t > 0$) do
      if( $|\mathcal{F}| = 1$ )then          /* a unique row singleton */
        $\mathcal{C}_v \leftarrow j \in \mathcal{J}_i;$ $\mathcal{R}_v \leftarrow i \in \mathcal{F};$
        $\mathcal{J}_k \leftarrow \mathcal{J}_k \setminus \{j\}$ $\forall k \in \mathcal{I}_j;$ $\mathcal{I}_j \leftarrow \emptyset;$ $v \leftarrow v + 1;$
      else
```

```
            $k \leftarrow 1$; $Max\_Tally(k, n, \mathcal{I}, \mathcal{C}, \mathcal{S}, s)$;
            $q \leftarrow |\{i : i \in \mathcal{I}_s, 0 \le |\mathcal{J}_i| \le 1\}|$;        /* Compute $t_{(s)}$ */
            $\mathcal{C}_r \leftarrow s$; $\mathcal{R}_r \leftarrow i \in \mathcal{F} \cap \mathcal{I}_s$;
            $\mathcal{J}_k \leftarrow \mathcal{J}_k \setminus \{s\}\ \forall k \in \mathcal{I}_s$; $\mathcal{I}_s \leftarrow \emptyset$; $r \leftarrow r + 1$; $\mathcal{F} \leftarrow \mathcal{F} \setminus \{i\}$;
            while ($q > 0$ & $r \le n$) do
              $q \leftarrow q - 1$;
              if( $q > 0$ ) then      /* include a spike into the agenda */
                $\mathcal{C}_r \leftarrow \mathcal{S}_i$; $\mathcal{R}_r \leftarrow i \in \mathcal{F}, |\mathcal{J}_i| = 0$;
                $\mathcal{F} \leftarrow \mathcal{F} \setminus \{i\}$; $t \leftarrow t - 1$; $r \leftarrow r + 1$;
              else if ($t > 0$ & $\Xi \neq \emptyset$) then
                $Recover\_2(n, u, \mathbf{B}, \mathcal{I}, \mathcal{C}, \mathcal{R}, \mathcal{S}, \Xi)$;
              end if
            end while    /* $q > 0$ */
          end if    /* $|\mathcal{F}| \neq 1$ */
          $\mathcal{F} \leftarrow \{i : |\mathcal{J}_i| = 1, 1 \le i \le n\}$;
        end while    /* $\mathcal{F} \neq \emptyset$ */
        $k \leftarrow \min\{|\mathcal{J}_i| : \mathcal{J}_i \neq \emptyset, 1 \le i \le n\}$;
        if($\{i : \mathcal{J}_i \neq \emptyset, 1 \le i \le n\} = \emptyset$ & $\Xi \neq \emptyset$) then $Recover\_2(n, u, \mathbf{B}, \mathcal{I}, \mathcal{C}, \mathcal{R}, \mathcal{S}, \Xi)$;
      end while
**end**.
```

Consider the permuted matrix illustrated in Figure 3. In this matrix, the unassigned rows and columns form the bump. Using the *Max_Tally* procedure, column 13 was selected for temporary removal from the bump. This removal introduced two row singletons, rows 8 and 18. Using the *Max_Tally* procedure, column 4 was selected to be assigned to $\mathcal{C}_6$, and row 8 was selected to be assigned to $\mathcal{R}_6$. This assignment introduced a new row singleton, row 13. Similarly, rows 20 and 13 were assigned to $\mathcal{R}_7$ and $\mathcal{R}_8$, where the corresponding columns and the order of placement were

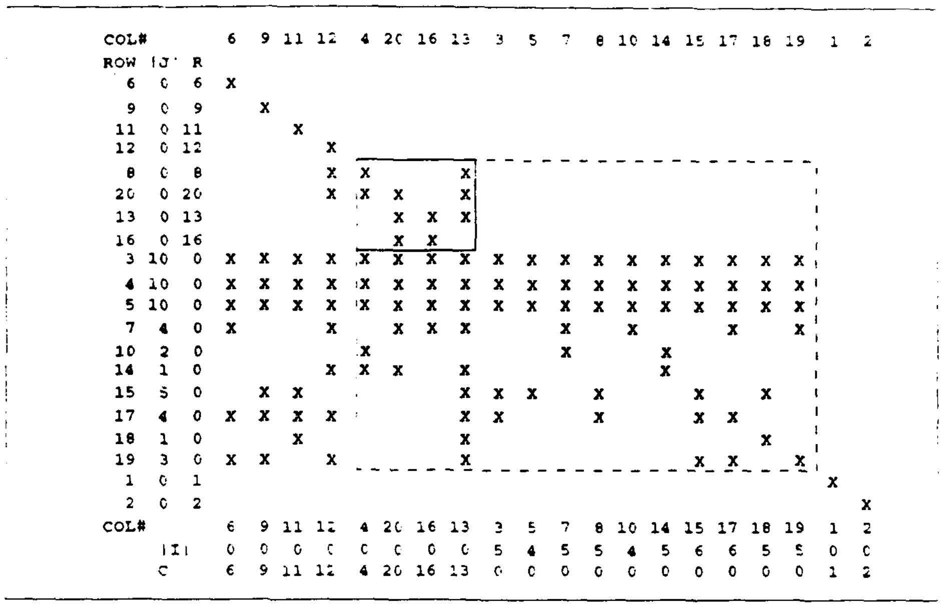
```
COL#           6   9  11  12   3   4   5   7   8  10  14  15  16  17  18  19  20  13   1   2
ROW |J|  R
  6   0   6    X
  9   0   9        X
 11   0  11            X
 12   0  12                X
  3  13   0    X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   S
  4  13   0    X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   S
  5  13   0    X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   S
  7   6   0    X           X               X       X           X   X       X   X   S
  8   1   0                X       X   1                                           S
 10   3   0                        X       X           X       3
 13   2   0                                                    X               X   S
 14   3   0                X       X                   X                       X   S
 15   5   0        X   X       X       X       X           X           X           S
 16   2   0                                                    X               X
 17   4   0    X   X   X   X   X               X           X       X               S
 18   1   0            X                                               X           S
 19   3   0    X   X       X                               X       X       X   2   S
 20   2   0                X       X                                           X   S
  1   0   1                                                                            X
  2   0   2                                                                                X
COL#           6   9  11  12   3   4   5   7   8  10  14  15  16  17  18  19  20  13   1   2
    |I|        0   0   0   0   5   7   4   5   5   4   5   6   6   6   5   5   8   0   0   0
      C        6   9  11  12   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   2
             K= 1    TK        0   1   0   0   0   0   0   0   0   0   1   0   0

             K= 2    TK            2                                   1
After including column 4 and row 8  into the agenda.
             K= 1    TK        0       0   0   0   0   0   0   0   0   1   0   1

             K= 2    TK                                                1       4
After including column 20 and row 20  into the agenda.
             K= 1    TK        0       0   0   0   0   1   0   2   0   1   0
```

Figure 5: Processing the bump after removing column 13 as a spike.

determined using the tally function. These assignments are illustrated in Figure 5. Since $t_1(16) = 2$, (i.e. the number of row singletons that have nonzero entries in column 16 is 2), one spike can be placed into the pivot agenda. Since column 13 was the only spike, including it into the pivot agenda decomposed the bump into two small blocks, as illustrated in Figure 6. Since minimum $|\mathcal{J}_i| = 1$ for the remaining rows, the *Row_Singl* procedure is invoked to assign these row singletons to the pivot agenda. The resulting matrix is shown in Figure 7. Since minimum $|\mathcal{J}_i|$ is now greater than one, the *Bump_Proc* procedure was invoked again, and the resulting matrix is illustrated in Figure 8. By processing the bump, the number of nonzero entries above the diagonal was reduced to only fifteen entries which are confined to four spikes in two blocks.



```
COL#           6   9  11  12   4  20  16  13   3   5   7   8  10  14  15  17  18  19   1   2
ROW |J| R
  6  0   6     X
  9  0   9         X
 11  0  11             X
 12  0  12                 X
  8  0   8                 X   X           X
 20  0  20                 X   X   X       X
 13  0  13                         X   X   X
 16  0  16                         X   X
  3 10   0     X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
  4 10   0     X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
  5 10   0     X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
  7  4   0     X           X       X   X   X           X       X           X       X
 10  2   0                     X                       X           X
 14  1   0                 X   X   X       X                       X
 15  5   0         X   X                   X   X   X       X           X       X
 17  4   0     X   X   X   X               X   X           X           X   X
 18  1   0             X                   X                                   X
 19  3   0     X   X       X               X                           X   X       X
  1  0   1                                                                             X
  2  0   2                                                                                 X
COL#           6   9  11  12   4  20  16  13   3   5   7   8  10  14  15  17  18  19   1   2
     |I|       0   0   0   0   0   0   0   0   5   4   5   5   4   5   6   6   5   5   0   0
     C         6   9  11  12   4  20  16  13   0   0   0   0   0   0   0   0   0   0   1   2
```

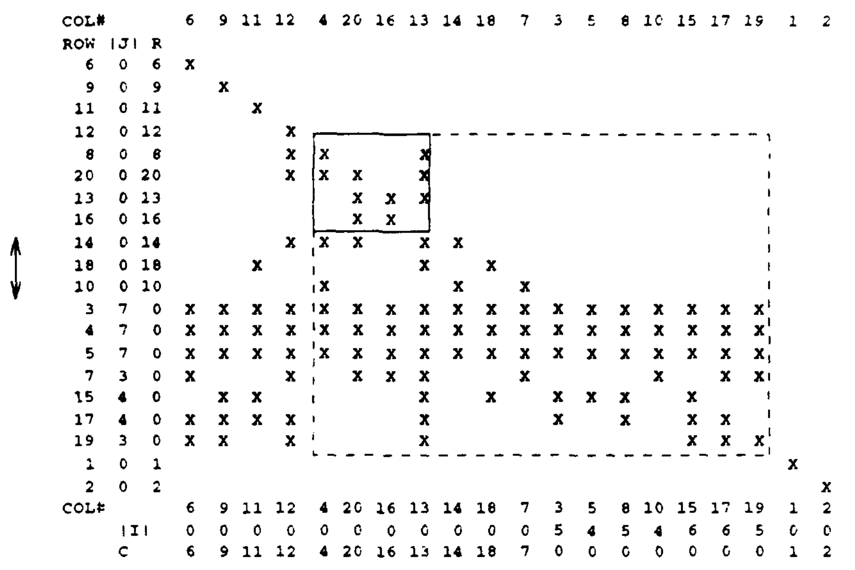
Figure 6: The matrix after decomposing a block.

6. Row Recovery

The above algorithms are designed to help produce a sparse factorization of the inverse of a linear programming basis. Due to finite precision of machines, round-off errors occur which may lead to a set of basic columns having a row of zeros. When this occurs, the classic Hellerman-Rarick P3 algorithm fails. In this section, we present the recovery algorithms which ensure that the prescribed lower triangular form is attainable, while introducing a replacement column having a single nonzero entry. The ingenuity is in the selection of the basic column to be replaced.

Let $\mathcal{Z} = \{i : 1 \leq i \leq n, \mathcal{J}_i = o\}$. Let $\mathbf{e}_i$ denote the i^{th} column of an identity matrix. Based on the structure of the original matrix, recovery can be done at three different stages of the algorithm.

The first case for recovery occurs when there exists at least one zero row, and all the remaining rows have been assigned. Since only $n - |\mathcal{Z}|$ pivots have been assigned,



```
COL#           6   9  11  12   4  20  16  13  14  18   7   3   5   8  10  15  17  19   1   2
ROW  |J|   R
   6   0   6   X
   9   0   9       X
  11   0  11           X
  12   0  12               X
   8   0   8               X   X           X
  20   0  20               X   X   X       X
  13   0  13                       X   X   X
  16   0  16                       X   X
  14   0  14               X   X   X       X   X
  18   0  18           X                   X       X
  10   0  10                   X               X       X
   3   7   0   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
   4   7   0   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
   5   7   0   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
   7   3   0   X           X       X   X   X           X               X       X   X
  15   4   0       X   X                   X       X       X   X   X       X
  17   4   0   X   X   X   X               X               X       X       X   X
  19   3   0   X   X       X               X                               X   X   X
   1   0   1                                                                           X
   2   0   2                                                                               X
COL#           6   9  11  12   4  20  16  13  14  18   7   3   5   8  10  15  17  19   1   2
     |I|       0   0   0   0   0   0   0   0   0   0   0   5   4   5   4   6   6   5   0   0
      C        6   9  11  12   4  20  16  13  14  18   7   0   0   0   0   0   0   0   1   2
```

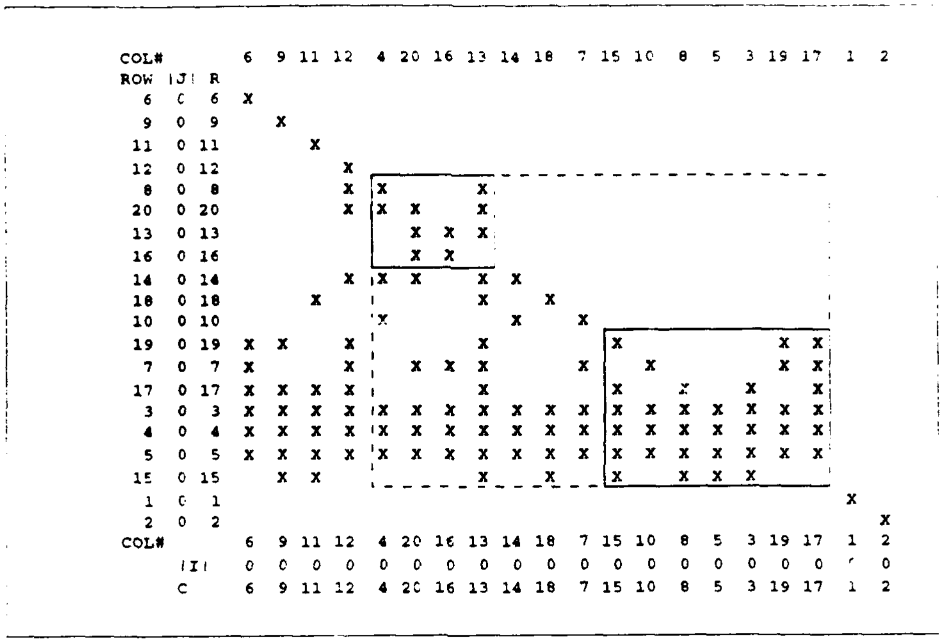
Figure 7: The matrix after applying the *Row_Singl* procedure for the second time.

$|\mathcal{I}_j| = 0$ for the remaining columns, and it is clear that these columns should be replaced. Recovery from this case can be mathematically described as follows:

```
Procedure : Recover_1
Input : $k, n, u, \mathbf{B}, \mathcal{I}, \Xi$.
Output : $u, \mathcal{C}, \mathcal{R}, \Xi$.
begin
    $\mathcal{C}_u \leftarrow j \in \{k : 1 \le k \le n, k \notin \mathcal{C}, \mathcal{I}_k = \phi\}$;
    $\mathcal{R}_u \leftarrow i \in \Xi$;  $\Xi \leftarrow \Xi \setminus \{i\}$;  $u \leftarrow u - 1$;  $k \leftarrow 0$;
    replace column $j$ of $\mathbf{B}$ by $\mathbf{e}_i$;
end.
```

This case is illustrated in Figure 9.

The second case for recovery occurs when there exists a zero row in the middle of processing the bump. The matrices permuted by the P3 algorithm have the interesting property of spikes appearing as properly nested sets, for more details see Arioli *et*



```
COL#           6   9  11  12   4  20  16  13  14  18   7  15  10   8   5   3  19  17   1   2
ROW  |J|   R
   6   0   6   X
   9   0   9       X
  11   0  11           X
  12   0  12               X
   8   0   8               X   X           X
  20   0  20               X   X   X       X
  13   0  13                       X   X   X
  16   0  16                       X   X
  14   0  14               X   X   X       X   X
  18   0  18           X                   X       X
  10   0  10                   X               X       X
  19   0  19   X   X       X               X                   X                   X   X
   7   0   7   X           X       X   X   X           X       X                   X   X
  17   0  17   X   X   X   X               X                   X       X       X       X
   3   0   3   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
   4   0   4   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
   5   0   5   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X   X
  15   0  15       X   X               X       X       X       X   X   X
   1   0   1                                                                           X
   2   0   2                                                                               X
COL#           6   9  11  12   4  20  16  13  14  18   7  15  10   8   5   3  19  17   1   2
     |I|       0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
       C       6   9  11  12   4  20  16  13  14  18   7  15  10   8   5   3  19  17   1   2
```

Figure 8: The matrix after processing the last bump.

al. [1]. A spike has been discovered for this recovery case, which corresponds to the outer most nested set, and that will never be placed into the pivot agenda. This spike is always replaced by the appropriate column of the identity matrix. Recovery from the second case can be mathematically described as follows:

```
Procedure : Recover_2
Input : $u$, $\omega$, $\mathbf{B}$, $\mathcal{I}$, $\mathcal{S}$, $\mathcal{Z}$.
Output : $u$, $\mathbf{B}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{S}$, $\mathcal{Z}$.
begin
    $\mathcal{C}_u \leftarrow j \leftarrow \mathcal{S}_1$;  $\mathcal{R}_u \leftarrow i \in \mathcal{Z}$;
    $\mathcal{S} \leftarrow \mathcal{S} \backslash \mathcal{S}_1$;  $\mathcal{Z} \leftarrow \mathcal{Z} \backslash \{i\}$;  $u \leftarrow u - 1$;
    replace column $j$ of $\mathbf{B}$ by $\mathbf{e}_i$;
end
```

This case is illustrated in Figure 10.

```
COL#           1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
ROW  |J|   R
   1   0   1   X   X   X
   2   0   2   X   X   X
   3   0   3   X   X   X
   4   0   4
   5   0   5
   6   0   6                       X
   7   0   7                           X
   8   0   8                           X   X
   9   0   9   X   X       X                   X
  10   0  10   X   X       X                       X
  11   0  11                   X                       X
  12   0  12   X   X   X   X   X   X   X   X   X   X   X   X
  13   0  13                                                   X
  14   0  14                                                       X
  15   0  15                                                           X
  16   0  16   X   X   X   X   X   X   X   X   X   X   X   X               X
  17   0  17   X   X   X   X   X   X   X   X   X   X   X   X                   X
  18   0  18   X   X   X   X   X   X   X   X   X   X   X   X                       X
  19   0  19   X   X   X   X   X   X   X   X   X   X   X   X                           X
  20   0  20   X   X   X   X   X   X   X   X   X   X   X   X                               X
COL#           1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
     |I|       0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
       C       1   2   3   0   0   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
```

Figure 9: An example of recovery case 1.

The third case for recovery occurs when there exists a zero row at the end of processing the bump. For this case, there are $|\Xi| - |S|$ unassigned columns and a spike has been discovered that will never be placed into the pivot agenda. Similar to the second case, this spike is always replaced by the appropriate column of the identity matrix using procedure *Recover_2*. Eventually, we will reach a situation similar to the first case, where procedure *Recover_1* will be invoked. This case is illustrated in Figure 11.

7. Summary and Conclusions

We believe that the ideas presented in this manuscript close a gap in the literature concerning the preassigned pivot agenda algorithms. All of these algorithms assume an input matrix which is nonsingular, that may not be the case every time these procedures are called. This manuscript presents a variant of P3 which results in a

```
COL#           7  13  11  16  17   4  20  19  18   3   5  10  15   6   1   2   8   9  12  14
ROW |J|  R
   7   0   7   X
  10   0  10   X   X
  11   0  11           X
  16   0  16               X
  17   0  17                   X
   6   0   6   X   X               X
  19   0  19   X           X       X   X
  13   0  13                           X   X   X                   S
  18   0  18   X           X       X       X   X                   S
   3   4   0   X   X       X   X   X   X   X   X   X   X   X   X   S
   4   4   0   X   X   X   X   X   X   X   X   X   X   X   X   X   S
   5   3   0   X   X       X   X   X   X   X   X       X   X   X   S
  15   1   0                                   X               X   S
   1   0   1                                                           X
   2   0   2                                                       S       X
   8   0   8                                                                   X
   9   0   9                                                                       X
  12   0  12                                                                           X
  14   0  14                                                                               X
  20   0   0
COL#           7  13  11  16  17   4  20  19  18   3   5  10  15   6   1   2   8   9  12  14
     |I|       0   0   0   0   0   0   0   0   0   2   3   3   4   0   0   0   0   0   0   0
       C       7  13  11  16  17   4  20  19  18   0   0   0   0   0   1   2   8   9  12  14
```

Figure 10: An example of recovery case 2.

reliable and efficient method to recover from all types of numerical instabilities. These ideas can be easily adapted for the algorithms of [1], [3], [4], [6], and [7].

References

[1] M. Arioli, I. S. Duff, N. I. Gould, and J. K. Reid, "Use of the P4 and P5 Algorithms for In-Core Factorization of Sparse Matrices," *SIAM Journal on Scientific and Statistical Computing*, Vol. 11, No. 5, pp. 913-927, 1990.

[2] I. S. Duff, "On Algorithms for Obtaining a Maximum Transversal," *ACM Transactions on Mathematical Software*, Vol. 7, No. 3, pp. 315-330, 1981.

[3] A. M. Erisman, R. G. Grimes, J. G. Lewis, and W. G. Poole Jr, "A Structurally Stable Modification of Hellerman-Rarick's P4 Algorithm for Reordering Unsy-

```
COL#           1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
ROW  |J|   R
   1   0   1   X   X   X           S
   2   0   2   X   X   X           S
   3   0   3   X   X   X           S
   4   0   0
   5   0   0
   6   0   0
   7   0   7                       S   X
   8   0   8                       S   X   X
   9   0   9           X   X       S       X   X
  10   0  10       X   X   X       S           X   X
  11   0  11                   X   S               X   X
  12   0  12   X   X   X   X   X   S   X   X   X   X   X   X
  13   0  13                                               X   X
  14   0  14                                                   X   X
  15   0  15                                                       X   X
  16   0  16   X   X   X   X   X   S   X   X   X   X   X   X           X   X
  17   0  17   X   X   X   X   X   S   X   X   X   X   X   X               X   X
  18   0  18   X   X   X   X   X   S   X   X   X   X   X   X                   X   X
  19   0  19   X   X   X   X   X   S   X   X   X   X   X   X                       X   X
  20   0  20   X   X   X   X   X   S   X   X   X   X   X   X                           X   X
COL#           1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
        |I|    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
         C     1   2   3   0   0   0   7   8   9  10  11  12  13  14  15  16  17  18  19  20
```

Figure 11: An example of recovery case 3.

metric Sparse Matrices," *SIAM Journal Numerical Analysis*, Vol. 2, pp. 369-385, 1985.

[4] B. Hattersley, and L. Mackley, "Construction of LU Factors of the Basis to Reduce Build-Up During Simplex Iterations," *Journal of the Operational Research Society*, Vol. 43, No. 5, pp. 507-518, 1992.

[5] E. Hellerman, and D. Rarick "Reinversion with the Preassigned Pivot Procedure," *Mathematical Programming*, Vol. 1, pp. 195-216, 1971.

[6] E. Hellerman, and D. Rarick, "The Partitioned Preassigned Pivot Procedure (P4)," in D. J. Rose and R. A. Willoughby, Eds. *Sparse Matrices and their Application*, pp. 67-76, Plenum Press, New York, 1972.

[7] J. K. Sankaran, "Some New Results Regarding Spikes and A Heuristic for Spike Construction," *Mathematical Programming*, Vol. 61, pp.171-195, 1993.

Distribution List

Dr. Neal D. Glassman
Program Manager
AFOSR/NM
110 Duncan Avenue, Suite 100
Bolling AFB, DC 20332−0001
(1 copy)

Marilyn J. McKee, Chief
Contract and Grant Administration Division
AFOSR/PKA
110 Duncan Avenue, Suite B115
Bolling AFB, DC 20332−0001
(6 copies)

Carol Voltner, Assistant Director
Office of Scientific Research
SMU
Dallas, TX 75275
(1 copy)